

**PRÁCTICA: 8. PL/SQL****OBJETIVOS:**

Conocer los fundamentos básicos de PL/SQL y aprender los componentes principales de un bloque PL/SQL.

Aprender a crear y usar procedimientos almacenados y utilizar cursores dentro de bloques PL/SQL.

**MATERIAL:**

ORACLE 9i versión para WINDOWS XP

**BIBLIOGRAFIA:**

ORACLE: Introduction to ORACLE 9i: PL/SQL. Student Guide. 2001.

O'REILLY: ORACLE PL/SQL Programming 2<sup>nd</sup> Edition. 2000.

**CONTENIDO:**

1. Introducción
2. Variables
  - 2.1. Declaración de variables
  - 2.2. Asignación de valores a variables
  - 2.3. Atributo %TYPE
  - 2.4. Variables BOOLEANAS
3. Órdenes ejecutables PL/SQL
  - 3.1. Comentarios PL/SQL
  - 3.2. Funciones PL/SQL
  - 3.3. Conversión de tipos
  - 3.4. Operadores
4. Entrada/Salida
  - 4.1. Entrada de datos
  - 4.2. Salida de datos
5. Órdenes SQL en PL/SQL
  - 5.1. SELECT
  - 5.2. INSERT, DELETE, UPDATE
6. Estructuras de control
  - 6.1. IF-THEN-ELSE
  - 6.2. LOOP
  - 6.3. FOR
  - 6.4. WHILE
7. Procedimientos y Funciones PL/SQL
  - 7.1. Procedimientos PL/SQL
  - 7.2. Funciones PL/SQL
8. Cursores
  - 8.1. Uso de cursores
  - 8.2. Cursores parametrizados
  - 8.3. Bucles de cursor FOR

Después de completar esta práctica, el alumno debería ser capaz de:

- Escribir bloques PL/SQL
- Manejar instrucciones SQL dentro de bloques PL/SQL.
- Escribir Procedimientos y Funciones PL/SQL
- Usar cursores PL/SQL.

## Esquema de Trabajo

Los ejemplos mostrados en esta práctica están basados en el siguiente esquema relacional:

DEPARTAMENTOS (codigo, nombre)

AREAS (codigo, nombre, departamento)

PROFESORES (codigo, apellido1, apellido2, nombre\_pila, activo, categoria, dedicacion, area)

ASIGNATURAS (siglas, nombre, creditos, curso, anualidad, clase, horas\_teoría, horas\_práctica, grupos\_teoría, grupos\_práctica, alumnos)

LOCALES (codigo, nombre, docente, capacidad, edificio, situacion)

GRUPOS (curso, clase, codigo, nombre)

DOCENCIA (id, curso, clase, grupo, siglas, profesor, local, dia, hora, periodicidad)

areas.departamento → departamentos

profesores.area → areas

docencia.curso, clase, grupo → grupos

docencia.profesor → profesores

docencia.local → locales

docencia.siglas → asignaturas

## 1. Introducción

PL/SQL es un lenguaje de programación estructurado. Es un lenguaje procedimental que amplía la funcionalidad de SQL, añadiendo estructuras habituales en otros lenguajes de programación, entre las que se encuentran:

- Variables y Tipos
- Estructuras de control
- Procedimientos y Funciones
- Tipos de Objetos y Métodos.

La unidad básica en PL/SQL es el bloque. Todos los programas PL/SQL están compuestos por bloques que pueden estar anidados. Un bloque PL/SQL está compuesto de tres partes principales:

- sección declarativa (opcional). Contiene las variables, constantes ...
- sección ejecutable (obligatoria). Contiene órdenes SQL para manipular datos de la base de datos y órdenes PL/SQL para manipular los datos del bloque
- sección de excepciones (opcional). Especifica las acciones a realizar en caso de error o cuando se producen excepciones en la ejecución.

La estructura general es:

```
[DECLARE
    variables, constantes, excepciones de usuario...]
BEGIN
    órdenes SQL
    órdenes PL/SQL
[EXCEPTION
    acciones a realizar al ocurrir un error]
END;
/
```

Para ejecutar un bloque PL/SQL siempre hay que colocar al final la barra /.

Podemos crear diferentes tipos de bloques:

- Bloques anónimos: Se construyen de forma dinámica y se suelen ejecutar una sola vez.
- Bloques nominados: Igual que los anónimos pero con una etiqueta que les da nombre.
- Subprogramas: Procedimientos, paquetes y funciones, almacenados en la base de datos y que se ejecutan en múltiples ocasiones. Los subprogramas se ejecutarán mediante una llamada.

- Disparadores (“Triggers”): Bloques nominados que se almacenan en la base de datos y que se ejecutan ante algún suceso.

Para poner nombre a un bloque se le pone una etiqueta antes del DECLARE encerrado por <<...>>. Por ejemplo para darle el nombre “Mi\_Bloque” a un bloque PL/SQL pondríamos:

```
<<Mi_Bloque>> DECLARE
...
BEGIN
...
END;
/
```

## 2. Variables

Las variables se definen en la sección declarativa de los bloques PL/SQL dónde también pueden inicializarse.

La asignación de nuevos valores a las variables puede hacerse en la parte ejecutable del bloque.

Pueden utilizarse para pasar valores como argumentos a subprogramas. Estas podrán ser de tipo IN (variable de entrada, OUT, variable de salida o INOUT, variable de entrada/salida).

También podrán utilizarse para almacenar valores devueltos o requeridos por una orden SQL.

Todas las variables tienen un tipo. Los posibles tipos de una variable son:

- Escalar. Almacenan un valor único. Son los mismos que los de las columnas de las tablas (VARCHAR2, NUMBER, DATE, CHAR, LONG, LONG\_RAW, BINARY\_INTEGER, LAW\_INTEGER) más el BOOLEAN
- Compuesto. Grupos de datos: tablas PL/SQL, registros...
- Puntero. Designan elementos de otros programas.
- LOB (Large Objects). Almacenan gran cantidad de información. Las variables de tipo LOB permiten almacenar datos no estructurados (imágenes, texto...) de hasta 4 GB de tamaño

## 2.1. Declaración de variables

Sintaxis:

```
<identificador> [CONSTANT] <tipo_de_dato> [NOT NULL] [{:= | DEFAULT  
<expresión>}];
```

Ejemplo:

```
DECLARE  
fecha DATE;  
dep_num NUMBER(2) NOT NULL := 10;  
ciudad VARCHAR2(10) := 'Ciudad Real';  
Km_a_milla CONSTANT NUMBER := 1.4;
```

Las variables declaradas como NOT NULL siempre deben ser inicializadas.  
La inicialización puede hacerse utilizando := o la palabra reservada DEFAULT.  
Si una variable no se inicializa contendrá el valor NULL.  
Las constantes deben ser inicializadas.

## 2.2. Asignación de valores a variables

Sintaxis:

```
<identificador> := <valor>;
```

## 2.3. Atributo %TYPE

El atributo %TYPE se utiliza para declarar una variable con el mismo tipo que una columna de una tabla o que otra variable definida anteriormente.

Sintaxis:

```
<identificador> {<tabla>.<columna> | <nombre_variable>}%TYPE;
```

Ejemplo:

```
var_nombre Empleados.nombre%TYPE;  
balance NUMBER;  
balance_minimo balance%TYPE := 10;
```

## 2.4. Variables BOOLEANAS

Las variable BOOLEANAS pueden tomar el valor TRUE, FALSE o NULL.

Las variables pueden combinarse mediante operadores lógicos (NOT, AND, OR).

Las expresiones pueden devolver valores BOOLEANOS utilizando operadores relacionales (<, <=...).

## 3. Órdenes Ejecutables

### 3.1. Comentarios en PL/SQL

Pueden añadirse comentarios al código. Estos comentarios pueden ser especificados con:

```
/*comentario
  más comentario */

-- comentario de línea
```

### 3.1. Funciones PL/SQL

Las funciones utilizables en SQL (LOWER, UPPER, INITCAP, CANCAT, SUBSTR, LENGTH, ROUND, TRUNC, MOD, MONTHS\_BETWEEN, ADD\_MONTHS, NEXT\_DAY, LAST\_DAY) excepto las de agrupamiento (ya que estas se aplican sobre una columna de una tabla).

### 3.2. Conversión de tipos

Existen funciones de conversión de tipos: TO\_CHAR, TO\_DATE, TO\_NUMBER

Sintaxis:

```
TO_CHAR (<valor>, <formato>)
TO_DATE(<valor>, <formato>)
TO_NUMBER(<valor>, <formato>)
```

### 3.3. Operadores

Los operadores en PL/SQL son los mismos que para SQL: Aritméticos, Lógicos, Concatenación, Paréntesis. Y además, existe el operador exponencial (\*\*).

## 4. Entrada/Salida

Los programas PL/SQL suelen realizar operaciones específicas sin interactuar con el operador. Sin embargo, existen algunas funciones que nos pueden ayudar a depurar programas y a interactuar con el usuario mostrando datos por pantalla y pidiendo datos al usuario.

### 4.1. Salida de datos

Para mostrar una cadena por pantalla podemos utilizar:

```
DBMS_OUTPUT.PUT_LINE(<cadena de caracteres>);
```

- Si los datos a mostrar no son cadenas puede utilizarse la función `TO_CHAR()` para transformarlo, y el operador `||` para concatenar. Con tipos de datos estándar (p. ej. Números) puede no ser necesario usar la función `TO_CHAR()`

El paquete `DBMS_OUTPUT` implementa una cola, en la cual se van almacenando los mensajes de salida. Si queremos que los mensajes aparezcan por pantallas tenemos que activar la opción `SERVEROUTPUT`:

```
SET ServerOutput ON;
```

### 4.2. Entrada de datos

Cuando trabajamos pidiendo datos al usuario es habitual especificar la opción `SET VERIFY OFF` para evitar que el sistema nos muestre el valor que tenía la variable antes y que nos confirme el nuevo valor que toma.

Para pedir datos al usuario se utiliza una variable de sustitución, dentro del código fuente del bloque PL/SQL, si esta variable no está inicializada, se le pedirá el valor al usuario:

```
SET ServerOutput ON;
SET VERIFY OFF;
DECLARE
  vv NUMBER :=&v;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Valor de v: '||vv);
END;
/
```

## 5. Órdenes SQL en PL/SQL

Podemos utilizar instrucciones SQL dentro de los bloques PL/SQL para recuperar datos de la base de datos o para actualizar los datos que contiene.

### 5.1. SELECT

Sintaxis:

```
SELECT <lista>
INTO {<variable>[, <variable>, ...] | <registro>}
FROM <tabla>
WHERE <condición>;
```

El SELECT almacenará los valores que obtenga en las variables indicadas tras INTO y en el mismo orden.

Es obligatorio incluir la cláusula INTO.

El SELECT debe prepararse para que sólo devuelva una fila.

Ejemplo:

```
DECLARE
    v_apellidos VARCHAR2(50);
    v_nombre VARCHAR2(30);
BEGIN
    SELECT apellidos, nombre
    INTO v_apellidos, v_nombre
    FROM Empleados
    WHERE id='5';
END;
```

### 5.2. INSERT, UPDATE, DELETE

La sintaxis no varía:

```
INSERT INTO tabla VALUES(...);
```

```
UPDATE tabla
SET valor = expresión
WHERE condición;
```

```
DELETE FROM tabla WHERE condición;
```

La única novedad es que en las expresiones y en las condiciones podemos utilizar variables PL/SQL.

## 6. Estructuras de control

### 6.1. Orden IF

El funcionamiento de la estructura IF es el habitual. Su Sintaxis es la siguiente:

```
IF <expresión1> THEN
    <Secuencia_ordenes1>;
[ELSIF <expresión2> THEN
    <Secuencia_ordenes2>;]
....
[ELSE
    <Secuencia_ordenesN>;]
END IF;
```

Ejemplo:

```
SET ServerOutput ON;
SET VERIFY OFF;
DECLARE
    v_num NUMBER := &v;
BEGIN
    IF v_num < 50 THEN
        DBMS_OUTPUT.PUT_LINE('Valor pequeño');
    ELSIF v_num < 100 THEN
        DBMS_OUTPUT.PUT_LINE('Valor mediano');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Valor grande');
    END IF;
END;
/
```

## 6.2. Orden LOOP

Los bucles LOOP son bucles que se ejecutan siempre, para salir de ellos tendremos que poner una instrucción de salida dentro del bucle.

Su Sintaxis es:

```
LOOP
[EXIT WHEN <condición>]
END LOOP;
```

Ejemplo:

```
SET ServerOutput ON;
SET VERIFY OFF;
DECLARE
    num NUMBER :=1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Valor: '|| num);
        num := num +1;
        EXIT WHEN num > 10;
    END LOOP;
END;
/
```

## 6.3. Orden FOR

Los bucles FOR se repiten un número determinado de veces.

Sintaxis:

```
FOR <contador> IN [REVERSE] <min>..<max> LOOP
...
END LOOP;
```

Ejemplo:

```
SET ServerOutput ON;
SET VERIFY OFF;
DECLARE
    num NUMBER;
BEGIN
    FOR num IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE('Valor: '|| num);
    END LOOP;
END;
/
```

## 6.4. Orden WHILE

Los bucles WHILE son iguales que en otros lenguajes de programación.

Sintaxis:

```
WHILE <condicion> LOOP
...
END LOOP;
```

Ejemplo:

```
SET ServerOutput ON;
SET VERIFY OFF;
DECLARE
    num NUMBER:=1;
BEGIN
    WHILE num <=10 LOOP
        DBMS_OUTPUT.PUT_LINE('Valor: '|| num);
        num := num + 1;
    END LOOP;
END;
/
```

## 7. Procedimientos y Funciones PL/SQL

Los procedimientos y las funciones son bloques PL/SQL con nombre que tienen un comportamiento especial. Los procedimientos son bloques que pueden ser invocados para realizar una tarea. Las funciones son bloques que realizan operaciones con los parámetros de entrada y devuelven el resultado de esas operaciones.

### 7.1. Procedimientos PL/SQL

Un procedimiento PL/SQL es similar a los procedimientos de otros lenguajes de programación. Un procedimiento es un bloque PL/SQL con nombre que tiene la misma estructura que los bloques anónimos.

La estructura general de un procedimiento PL/SQL es:

```
CREATE OR REPLACE PROCEDURE nombre (
    param [IN | OUT | INOUT] <tipo> [, param [IN|OUT|INOUT] <tipo>,...] {IS|AS}
[DECLARE]
BEGIN
    <codigo del procedimiento>
[EXCEPTION]
END;
```

Cuando se crea un procedimiento, éste se compila en primer lugar y queda almacenado en la base de datos de forma compilada. El código compilado puede ser posteriormente utilizado por cualquier bloque PL/SQL.

Para modificar un procedimiento creado debemos reemplazarlo por el nuevo volviendo a compilarlo añadiendo las palabras clave **OR REPLACE**. Podemos eliminar un procedimiento mediante la orden **DROP PROCEDURE <nombre>**

Ejemplo:

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE PROCEDURE Muestra (Cad IN Varchar(20)) AS
BEGIN
    DBMS_OUTPUT.PUT_LINE (Cad);
END;
```

```
DECLARE
    Mens := &Mensaje
BEGIN
    Muestra(Mens);
END Muestra;
```

## 7.2. Funciones PL/SQL

Las funciones son iguales que los procedimientos pero además devuelven un valor, por lo que la llamada a una función debe realizarse dentro de una expresión.

La estructura general de una función es:

```
CREATE OR REPLACE FUNCTION nombre (
    param [IN | OUT | INOUT] <tipo> [, param [IN|OUT|INOUT] <tipo>,...]
    RETURN <expresión> {IS|AS}
[DECLARE]
BEGIN
    <codigo del procedimiento>
    RETURN <valor>
[EXCEPTION]
END;
```

La orden **RETURN** dentro de una función devuelve el valor que la función debe devolver, el cual se convierte al tipo especificado en la cabecera de la función. Puede haber más de una instrucción **RETURN**, pero solo se ejecutará la primera que se encuentre dentro de la lógica del programa.

Ejemplo:

```
SET SERVEROUTPUT ON;
CREATE OR REPLACE FUNCTION Factorial (N IN NUMBER) RETURN NUMBER
AS
BEGIN
    IF N<0 THEN
        Return -1;
    ELSIF N=0 THEN
        Return 1;
    ELSE
        Return N*Factorial(N-1);
    END IF;
END Factorial;
```

```
DECLARE
    v NUMBER := &valor;
    f NUMBER;
BEGIN
    f := Factorial(v);
    DBMS_OUTPUT.PUT_LINE ('Factorial de ' || v || ' = ' || f);
END;
/
```

## 8. Cursores

Los cursores son punteros a unas zonas de memoria llamadas áreas de contexto en las cuales se almacena información acerca del resultado de una consulta, así como el conjunto de registros procesados y no procesados.

Mediante este puntero (cursor), un programa PL/SQL puede controlar el área de contexto y manejar el conjunto de registros devueltos por una consulta.

### 8.1. Uso de cursores

Para poder utilizar un cursor debemos hacer cuatro pasos:

1. Declaración del cursor
2. Apertura del cursor
3. Recogida de los resultados en variables PL/SQL
4. Cierre del cursor

Para declarar un cursor se utiliza la siguiente sintaxis:

```
CURSOR <nombre_cursor> IS <orden_SELECT>
```

Para poder abrir un cursor utilizaremos:

```
OPEN <nombre_cursor>
```

Para recoger los datos que devuelve un cursor utilizaremos la orden FETCH de la siguiente manera:

```
FETCH <nombre_cursor> INTO {<lista_variables>|<registro>}
```

Después de utilizar el FETCH se incrementa el puntero del conjunto activo para que apunte al siguiente registro, de tal manera que en un bucle, cada FETCH devolverá filas sucesivas del conjunto activo. El atributo de cursores %NOTFOUND se utiliza para saber cuando se ha terminado de recorrer el conjunto activo.

Para cerrar un cursor utilizaremos:

```
CLOSE <nombre_cursor>;
```

Los cursores tienen cuatro atributos:

%FOUND devuelve TRUE si la última instrucción FETCH devolvió una fila

%NOTFOUND es la contraria a %FOUND

%ISOPEN nos indica si el cursor está abierto

%ROWCOUNT nos dice el número de registros extraídos por el cursor hasta el momento.

Ejemplo:

```
SET SERVEROUTPUT ON;
```

```
DECLARE
```

```
    v_CODIGO  AREAS.CODIGO%TYPE;
```

```
    v_NOMBRE  AREAS.NOMBRE%TYPE;
```

```
    CURSOR c_AREAS IS
```

```
        SELECT CODIGO, NOMBRE
```

```
        FROM AREAS
```

```
        ORDER BY CODIGO;
```

```
BEGIN
```

```
    OPEN c_AREAS;
```

```
    LOOP
```

```
        FETCH c_AREAS INTO v_CODIGO, v_NOMBRE;
```

```
        EXIT WHEN c_AREAS%NOTFOUND;
```

```
        DBMS_OUTPUT.PUT_LINE(v_CODIGO || ' - ' || v_NOMBRE);
```

```
    END LOOP;
```

```
    CLOSE c_AREAS;
```

```
END;
```

```
/
```

## 8.2. Cursores parametrizados

Permiten utilizar la orden OPEN para enviar las variables de acoplamiento de un cursor. Por ejemplo, el siguiente código crea un cursor para las áreas cuyo código se pasa por parámetro en la orden OPEN:

```
SET SERVEROUTPUT ON;
DECLARE
    v_CODIGO AREAS.CODIGO%TYPE;
    v_NOMBRE AREAS.NOMBRE%TYPE;

    CURSOR c_AREAS (v_COD AREAS.CODIGO%TYPE) IS
        SELECT CODIGO, NOMBRE
        FROM AREAS
        WHERE CODIGO = v_COD
        ORDER BY CODIGO;

BEGIN
    OPEN c_AREAS ('ATC');
    LOOP
        FETCH c_AREAS INTO v_CODIGO, v_NOMBRE;
        EXIT WHEN c_AREAS%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(v_CODIGO || ' - ' || v_NOMBRE);
    END LOOP;
    CLOSE c_AREAS;

END;
/
```

### 8.3. Bucles de cursor FOR

Ya hemos visto como podemos utilizar los cursores para recorrer los resultados de una consulta mediante el uso de bucles LOOP y podemos realizar las mismas operaciones utilizando bucles WHILE. Cuando utilizamos bucles FOR podemos usar un procesamiento mucho más sencillo con los bucles de cursor FOR, los cuales abren, recuperan los datos y cierran el cursor de forma implícita, haciendo su uso más sencillo.

Su sintaxis es la siguiente:

```
FOR <variable> IN <cursor> LOOP
<codigo>
END LOOP;
```

Ejemplo:

```
SET SERVEROUTPUT ON;
DECLARE
    CURSOR C_PROFES IS
        SELECT nombre_pila
        FROM profesores
        ORDER BY codigo;

BEGIN
    FOR VP IN C_PROFES LOOP
        DBMS_OUTPUT.PUT_LINE ('Nombre: ' || VP.nombre_pila);
    END LOOP ;
END ;
/
```

## EJERCICIOS PRÁCTICA 8 –PL/SQL

Para realizar estos ejercicios es necesario recordar la estructura del esquema de la base de datos ACADEM empleada en prácticas anteriores:

DEPARTAMENTOS (codigo, nombre)

AREAS (codigo, nombre, departamento)

PROFESORES (codigo, apellido1, apellido2, nombre\_pila, activo, categoria, dedicacion, area)

ASIGNATURAS (siglas, nombre, creditos, curso, anualidad, clase, horas\_teoría, horas\_practica, grupos\_teoría , grupos\_practica, alumnos)

LOCALES (codigo, nombre, docente, capacidad, edificio, situacion)

GRUPOS (curso, clase, codigo, nombre)

DOCENCIA (id, curso, clase, grupo, siglas, profesor, local, dia, hora, periodicidad)

areas.departamento → departamentos

profesores.area → areas

docencia.curso, clase, grupo → grupos

docencia.profesor → profesores

docencia.local → locales

docencia.siglas → asignaturas

1. Escribir un bloque PL/SQL que calcule la media de dos números dados por el usuario.
2. Escribir un bloque PL/SQL que pida dos identificadores de profesores y determine cual de los dos está más cerca de cero o si son iguales
3. Escribir un bloque PL/SQL que convierta una determinada cantidad expresada en segundos a horas, minutos y segundos.
4. Mostrar el contenido de una variable que contenga la capacidad total de todas las aulas del edificio de Informática (EUI)
5. Insertar una fila en la tabla local con la capacidad total de los espacios de magisterio ('EGB') (poner como nombre de edificio 'TOT', inventarse el contenido del resto de las columnas). Obtener la capacidad máxima de entre todas las clases del edificio de Informática ('EUI'). Restar, de la capacidad de la fila insertada, el último valor obtenido actualizar la fila correspondiente en la tabla. Eliminar las filas del edificio 'TOT'.
6. Cree una función que calcule la media de dos números. Compruebe su funcionamiento.
7. Mostrar, ordenados por el código, el nombre de todos los profesores, utilizando un cursor.
8. Utilizando un cursor muestre las asignaturas cuyo ID sea par.